Annotation for the Deep Web

Siegfried Handschuh and Raphael Volz, *University of Karlsruhe*

Steffen Staab, *University of Karlsruhe and Ontoprise GmbH*

One of the core challenges of the Semantic Web is to create metadata by mass collaboration—by combining semantic content created by a large number of people. To attain this objective, researchers have developed several approaches[1–3] to deal with creating manual or semiautomatic metadata from existing information. However, most of these approaches build on the assumption that the information sources are static, such as static HTML pages or books in a library (scenarios A and B in Table 1).

One of the core challenges of the Semantic Web is to create metadata by mass collaboration. A solution to this problem is a technique called deep annotation, which uses three elements of information—the information itself, its structure, and its context—to derive mappings.

Today, however, a large percentage of Web pages are dynamic. Estimates about the ratio of static to dynamic pages based on Web pages actually crawled by search engines typically conclude that dynamic Web pages outnumber static ones by 100 to 1. Manually annotating every single dynamic Web page generated—for example, from a database that contains a catalog of books—would be tedious. It would be better to annotate the database so that it is reusable for site-specific Semantic Web purposes.

To achieve this objective, several approaches provide for constructing wrappers by explicitly defining HTML or XML queries or by learning such definitions from examples[4] (scenario C in Table 1). See the "Related Work" sidebar for a discussion of related approaches. It has become possible with these approaches to create metadata manually for a set of structurally similar Web pages. These approaches offer the advantage of not requiring the database owner's cooperation. But the drawback to the wrapper approach is that the correct scraping of metadata depends largely on data layout rather than on the structures underlying the data.

We assume that many sites will participate in the Semantic Web and will share information. Such Web sites might present their information only as HTML pages. Although the Web site administrator might be reluctant to provide the underlying data so that it conforms to a client's ontology, the administrator might conveniently describe its current structure on the very same Web pages. Thus, these sites could give users the option of using the information itself, the information structure, or the information context to create mappings to other information structures. This process is called *deep annotation* (scenario D in Table 1). Users could then exploit these mappings to query the database underlying a Web site to retrieve semantic data. This process combines the capabilities of conventional annotation with the full capabilities of the databases.

Table 1 summarizes the different approaches, A to D. Furthermore, it shows another parallel between scenario A and the deep-annotation scenario D: In scenario A, the annotator can choose between embedding the metadata created in the annotation process into the information proper (through an HTML metatag) or keeping it remote. Correspondingly for deep annotation, the two choices boil down to storing the created mappings at the server or at the client side.

Uses for deep annotation

Countless potential application areas for deep annotation could be relevant for a large and quickly growing number of sites that target cooperation. For example, scientific databases are frequently built to foster cooperation among researchers. Medline, Swissprot, and EMBL are just a few examples of scientific databases on the Internet. Many estimate that more than 500 databases are freely accessible in the bioinformatics community alone.

Such databases are frequently hard to understand,

Table 1. Static versus dynamic sites.

Web site	Cooperative owner	Uncooperative owner
Static	(A) Embedded or remote metadata by conventional annotation	(B) Remote metadata by conventional annotation
Dynamic	(D) Deep annotation with server- or client-side mapping rules	(C) Wrapper construction, remote metadata

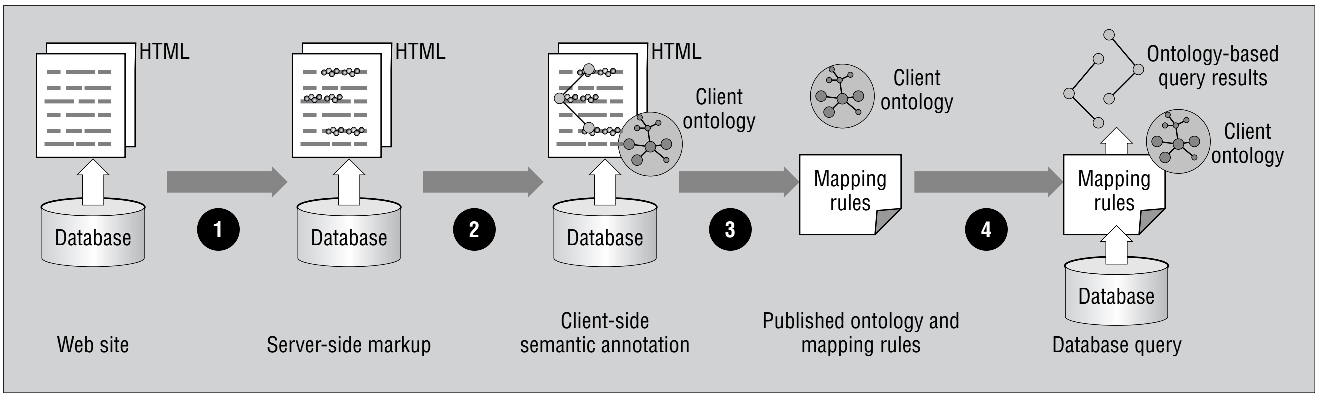


Figure 1. The four-step process of deep annotation.

and it is often difficult to evaluate whether a database table named "species" is equivalent to a table named "organism" in another database. Exploiting the information found in concrete tuples might help. But whether the "leech" considered as entry to an "organism" is actually the animal or the plant might be much easier to tell from the context in which it is presented than from the concrete database entry, which might resolve to "plant" or "animal" only through several joins.

Another case for deep annotation is in supply-chain scenarios. Car manufacturers frequently outsource the problem of providing mappings to their databases, and they typically offer their suppliers only a portal with HTML pages. Suppliers must then either retype information or replicate it using wrappers. If manufacturers provided information structures on their portals, a one-time deep-annotation process could easily create the mapping for new suppliers.

In addition to direct access to HTML pages of news stories or market research reports, commercial information providers frequently offer syndication services. Integrating such services into a customer's portal is typically an expensive, manual programming effort that could be reduced by a deep-annotation process that defines the content mappings.

Perhaps the most significant way deep annotation can serve the Internet is through community Web portals. A recent example based on Semantic Web technology is www.ontoweb.org. A community portal serves a community's information needs by letting members contribute and share information. Some are even designed to deliver semantic information back to their community as well as to the outside world.

The primary objective of a community setting up a portal will always be to provide human viewers with access to pertinent information. However, given the appropriate tools, portal designers could better serve their members by easily providing deep-annotation pages to their members with the appropriate information content, information structure, and information context.

Architecture

The process of deep annotation consists of several steps, shown in Figure 1. A Web site must be driven by an underlying relational database, with the search results derived from a server-side markup structure. The mapping rules between database and client ontology must be available with the results derived from the database. The entire deep-annotation process consists of four main steps:

1. The database owner produces server-side Web page markup according to the database's information structures.
2. The annotator produces client-side annotations that conform to the client ontology and the server-side markup.
3. The annotator publishes the client ontology and the mapping rules derived from the annotations.
4. The querying party loads the second party's ontology and mapping rules and uses them to query the database through the Web-service API.

In this process, a single person could be the database owner, the annotator, and the querying party. In other words, the deep-annotation process can work just as well with local or internal data repositories.

To use deep annotation with our community Web portal, for example, the annotator would annotate an organization entry from www.ontoweb.org according to the annotator's own ontology. Then the annotator would use the ontology and mapping to instantiate the syndication services by regularly querying all recent entries for titles to match the list of topics.

As Figure 2 shows, our architecture for deep annotation consists of three major pillars that correspond to the three different process roles: database owner, annotator, and querying party. At the Web site, we assume that there is an underlying database and a server-side scripting environment to create dynamic Web pages. The Web site might also provide a Web service interface to third-party entities that want to query the database directly.

Our annotator uses an extended version of OntoMat-Annotizer to create relational metadata that corresponds to a given client ontology. The extended OntoMat-Annotizer accounts for problems that might arise from generic annotations required by deep annotation. With the help of OntoMat-Annotizer, we can create mapping rules from such anno-

Related Work

Deep annotation, as we present it in this article, is a multidisciplinary field much like the Semantic Web research community. There are therefore several communities that have contributed to reaching deep annotation. So far, we have identified those related to information integration, mapping frameworks, wrapper construction, and annotation.

Information integration

Research surrounding information integration seeks to provide an algebra to translate information between different structures. In this field, underlying algebras are used to provide compositionality of translations as well as a sound basis for query optimization.[1,2] Our objective has not been to provide a flexible, scalable integration platform. Rather, the purpose of deep annotation lies in providing a flexible framework for creating the translation descriptions that we can then exploit with an integration platform.

Mapping and merging frameworks

We can distinguish approaches for mapping or merging ontologies or database schema mainly along three lines: discovery,[3–6] mapping representation,[7–9] and execution.[10] Generally speaking, there are researchers[11] whose approach is close to our own because it handles the complete mapping process involving the three process steps. What distinguishes deep annotation from all these approaches, however, is that for the initial discovery of overlaps between different ontologies, they all depend on lexical agreement of part of the two database schemata. Deep annotation depends only on the user understanding the presentation.

Wrapper construction

Methods associated with wrapper construction can achieve many objectives similar to our own. Some researchers have designed wrappers to allow for construction by explicitly defining HTML or XML queries or by learning such definitions from examples. The wrapper approaches have the advantage of not requiring cooperation with the database owner. However, the disadvantage is that the correct scraping of metadata depends to a large extent on data layout rather than on the structures underlying the data.

Furthermore, when the system provides definitions explicitly, the user must cope directly with layout constraints. When the system learns definitions, the user must annotate multiple Web pages to derive correct definitions. Also, these approaches do not map to ontologies. They typically map to lower-level representations—nested string lists from which the conceptual descriptions must be extracted. We have integrated a wrapper-learning method, called Amilcare, into our OntoMat-Annotizer.[12] The process of bridging between wrapper construction and annotation is described elsewhere.[13]

Annotation proper

Finally, we must consider annotation proper as part of deep

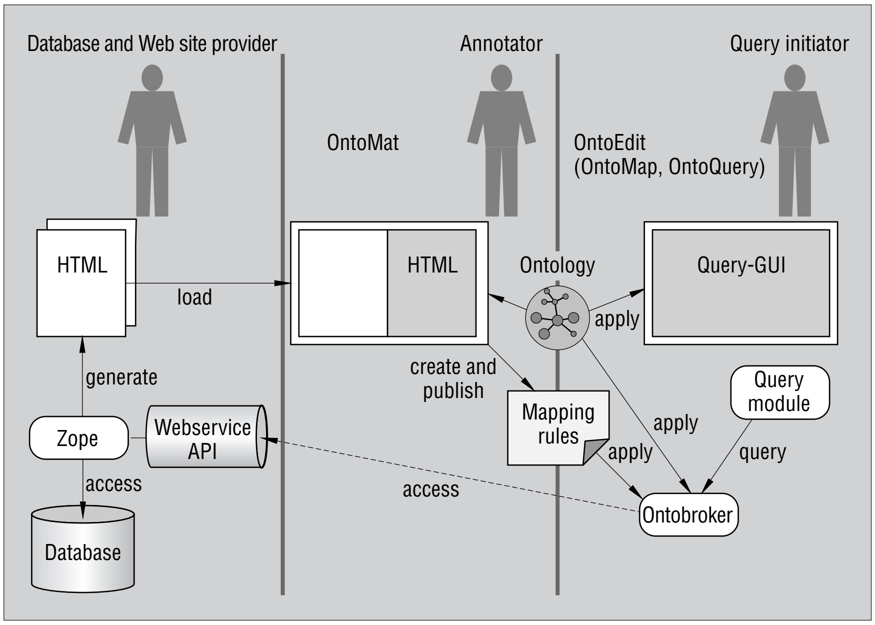


Figure 2. An architecture for deep annotation.

tations that are later exploited by an inference engine.

The querying party uses a corresponding tool to visualize the client ontology, compile a query from the client ontology, and investigate the mapping. We use OntoEdit for these three purposes. OntoEdit lets us investigate, debug, and change given mapping rules. It also lets us integrate and exploit the Ontobroker inference engine.[5]

Markup and annotation

The goal of the mapping process is to give interested parties access to the source data. Instead of offering the material directly, we provide pointers to the underlying data sources in the annotations. That is, we specify which database columns provide the data for certain instance attributes depicted on the Web site and thus combine the capabilities of databases with conventional annotation.

All information, including the structure of all tables involved in a Web site query, must be published so that users can retrieve data. We specify the database representation using a dedicated deep-annotation ontology, which we instantiate to describe the physical structure of the database element that will help decipher the Web site query results. We therefore can publish the structure of all tables involved in a Web site query. For example, the markup shown in Figure 3 is part of the HTML head of the Web page presented in Figure 4.

In Figure 3, the property *accessService* of the `<DB>` class represents the link to a service that allows anonymous database access. We rely on a Web service to host the database access to avoid local protocol issues, because most standard database connections are made through sockets on proprietary ports. As Figannotation. In this case, we have inherited the principal annotation mechanism for creating relational metadata.[14] The interested reader will find an elaborate comparison of annotation techniques in a forthcoming book on annotation.[15]

References

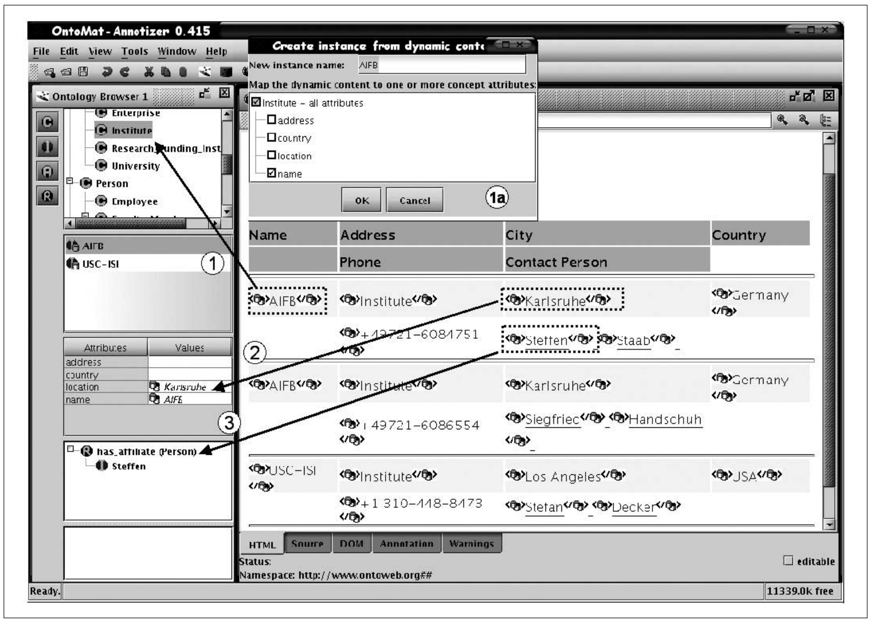
1. Y. Papakonstantinou and V. Vassalos, "Architecture and Implementation of an XQuery-Based Information Integration Platform," *IEEE Data Eng. Bulletin*, vol. 25, no. 1, 2002, pp. 18–26.
2. G. Wiederhold, "Intelligent Integration of Information," *Proc. 1993 ACM SIGMOD Int'l Conf. Management of Data*, 1993, ACM Press, pp. 434–437.
3. S. Bergamaschi et al., "Semantic Integration of Heterogeneous Information Sources," *Special Issue on Intelligent Information Integration, Data & Knowledge Eng.*, Elsevier Science, 2001, pp. 215–249.
4. W. Cohen, "The WHIRL Approach to Data Integration," *IEEE Intelligent Systems*, vol. 13, no. 5, Sept. 1998, pp. 20–23.
5. N.F. Noy and M.A. Musen, "PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment," *Proc. AAAI/IAAI 2000*, AAAI Press / MIT Press, 2000, pp. 450–455.
6. E. Rahm and P. Bernstein, "A Survey of Approaches to Automatic Schema Matching," *Very Large Databases J.*, vol. 10, no. 4, 2001, pp. 334–350.
7. J. Madhavan, P.A. Bernstein, and E. Rahm, "Generic Schema Matching with Cupid," *Proc. 27th Int'l Conf. Very Large Databases* (VLDB 01), Morgan Kaufmann, 2001, pp. 49–58.
8. P. Mitra, G. Wiederhold, and M. Kersten, "A Graph-Oriented Model for Articulation of Ontology Interdependencies," *Proc. Conf. Extending Database Technology*, Konstanz, 2000, pp. 86–100.
9. J.Y. Park, J.H. Gennari, and M.A. Musen, *Mappings for Reuse in Knowledge-Based Systems*, tech. report SMI-97-0697, Stanford Univ., 1997.
10. T. Critchlow, M. Ganesh, and R. Musick, "Automatic Generation of Warehouse Mediators Using an Ontology Engine," *Proc. 5th Int'l Workshop Knowledge Representation Meets Databases* (KRDB 98), Swiss Life, pp. 8.1–8.8.
11. A. Maedche et al., "MAFRA: A Mapping Framework for Distributed Ontologies," *Proc. EKAW 2002*, Springer-Verlag, 2002, pp. 235–250.
12. F. Ciravegna, "Adaptive Information Extraction from Text by Rule Induction and Generalisation," *Proc. 17th Int'l Conf. Artificial Intelligence*, Morgan Kaufmann, 2001, pp. 1251–1256.
13. S. Handschuh, S. Staab, and F. Ciravegna, "S-CREAM: Semiautomatic CREAtion of Metadata," *Proc. EKAW 2002*, Springer-Verlag, 2002, pp. 358–372.
14. S. Handschuh and S. Staab, "Authoring and Annotation of Web Pages in CREAM," *Proc. 11th Int'l World Wide Web Conf.*, ACM Press, 2002, pp. 462–473.
15. S. Handschuh and S. Staab, eds., *Annotation in the Semantic Web*, IOS Press, 2003.

```
<!--
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns ="http://annotation.semanticweb.org#deepanno">
 <DB rdf:ID="OntoSQL">
  <accessService
    rdf:resource="www.ontoweb.org/database_access.wsdl"/>
 </DB>
 <Table rdf:ID="Person">
  <name>Person</sqlName>
  <inDatabase rdf:resource="#OntoSQL" />
  <hasColumns rdf:parseType="Collection">
    <PrimaryKey rdf:ID="Person.ID"
      name="ID" type="int" />
    <Column name="FIRSTNAME" type="varchar"/>
    <Column name="LASTNAME" type="varchar"/>
  </hasColumns>
 </Table>
 <Table rdf:ID="Organization">
  <name>Organization</name>
  <inDatabase rdf:resource="#OntoSQL" />
  <hasColumns rdf:parseType="Collection" />
    <PrimaryKey rdf:ID="Organization.ID"
      name="ID" type="int" />
    <Column name="ORGNAME" type="varchar"/>
    <Column name="LOCATION" type="varchar"/>
    ...
  </hasColumns>
 </Table>
 <Table rdf:ID="PersonOrg">
  <name>Person_Org<name>
  <inDatabase rdf:resource="#OntoSQL" />
  <hasColumns rdf:parseType="Collection" />
    <PrimaryKey name="PERSONID" type="int">
      <references rdf:resource="#Person.ID"/>
    </PrimaryKey>
    <PrimaryKey name="ORGID" type="int">
      <references rdf:resource="#Organization.ID"/>
    </PrimaryKey>
  </hasColumns>
 </Table>
</rdf:RDF>
-->
```

Figure 3. Markup for a dedicated deep-annotation ontology.

ure 5 shows, we place the Web site query itself—used to retrieve the data from a particular source—in the header of the page. The header contains the SQL query and is associated with a name to distinguish between queries.

The structure of the query result must be published by means of column groups. Each column group must have at least one identifier, which is used in the annotation process to distinguish individual instances and detect their equivalence. Because the database keys are



Name	Address	City	Country
	Phone	Contact Person	
AIFB	Institute	Karlsruhe	Germany
	+49721-6084751	Steffen Staab	
AIFB	Institute	Karlsruhe	Germany
	+49721-6086554	Siegfried Handschuh	
USC-ISI	Institute	Los Angeles	USA
	+1 310-448-8473	Stefan Decker	

Figure 4. A screen shot of the deep-annotation system with OntoMat-Annotizer.

```
<!--
<rdf:RDF
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
 xmlns ="http://annotation.semanticweb.org#deepanno">
  <Query rdf:ID="Q1">
    <source rdf:resource="#OntoSQL" />
    <hasResultColumns rdf:parseType="Collection">
      <ColumnGroup rdf:about="#g1" />
      <ColumnGroup rdf:about="#g2" />
    </hasResultColumns>
    <sql>
    SELECT  Person.*, Person_Org.Orgid, Organization.*
    FROM    Person, Organization, Projekt_Org
    WHERE   Person.ID = Projekt_Org.PERSONID
            AND Organization.ID = Projekt_Org.ORGID
    </sql>
  </Query>
  <Columngroup rdf:ID="#g1">
    <prefix
      rdf:resource="http://www.ontoweb.org/person/">
    <hasColumns rdf:parseType="Collection">
      <Identifier name="Id" />
      <Column name="Firstname" />
      <Column name="Lastname" />
    </hasColumns>
  </Columngroup>
  <Columngroup rdf:ID="#g2">
    <prefix
      rdf:resource="http://www.ontoweb.org/org/">
    <hasColumns rdf:parseType="Collection">
      <Identifier name="OrganizationId" />
      <Column name="Orgname" />
      <Column name="Location" />
    </hasColumns>
  </Columngroup>
 </rdf:RDF>
-->
```

Figure 5. Markup for the SQL query found in the page header.

local to the respective table—while the Semantic Web has global identifiers—we must establish appropriate prefixes. The prefix ensures that we can detect the equality of instance data generated from multiple queries if the person maintaining the Web chooses the same prefix for each occurrence of that ID in a query. Eventually, the database keys translate to instance identifiers through the following pattern:

$$< prefix > [key_i - name = key_i - value]$$

Whenever we use parts of the query results in the dynamically generated Web page, we surround the generated content with a tag that carries information about which column represents the used value. To stay compatible with HTML, we use the <span> tag as an information carrier. The actual information is represented in the attributes of <span>, as in the following tag:

```
<span qresult="q1"
column="Orgname">AIFB</span>
```

The annotation tool then interprets such span tags and uses them in the mapping process. An annotation in our context is a set of instantiations related to an ontology and referring to an HTML document. We distinguish

- Instantiations of DAML+OIL classes
- Instantiated properties from one class instance to a data type instance
- Instantiated properties from one class instance to another class instance

For deep annotation, we distinguish between *generic* and *literal* annotations. In a literal annotation, the piece of text might stand for itself. In a generic annotation, we consider a piece of text that corresponds to a database field to be a placeholder. That is, we must generate a variable for such an annotation. The variable can have multiple relationships that allow for the description of general mapping rules. For example, the concept institute in the client ontology could correspond to one generic annotation for the organization identifier in the database.

Our user interface supports an annotation process of server-side markup when the user opens a Web page. The browser then handles the server-side markup and provides graphical icons on the page so the user can identify values that come from the database. The user selects one of the server-side markups to create a new generic instance and map the database field to a generic attribute. Along with the generic instance, the system stores the database information necessary to query the database in a later step.

When the user drags a server-side markup onto an ontology concept, the system generates a new generic class instance. The application displays a dialog for selecting the instance name and the attributes to which the database value is to be mapped. The system preselects attributes that resemble the column name. If the user clicks "OK," the system performs several checks and then creates the new generic instance. Generic instances, which appear with a database symbol in their icon, store the information about the database query and the unique identifier pattern.

The server-side markup contains the reference to the query, the column, and the value. The system obtains the identifier pattern from the reference to the query description and the column group. The markup used to create the instance defines the identifier pattern for the generic instance. The system uses the identifier pattern when generating instances from the database.

For example, if a user selects the server-side markup AIFB and drops it on the concept institute, the content of the markup is <span qresult="q1" column="Orgname">AIFB</span>, which creates a new generic instance with a reference to the query ql. The dialog-based choice for the instance name AIFB assigns the generic attribute name with the database column Orgname, which defines the identifier pattern of the generic instance as in the following fictitious URL: www.ontoweb.org/org/OrganizationID =$OrganizationID. The OrganizationID tag is the name of the database column in query q1 that holds the database key.

To create a generic attribute instance, the user simply drops the server-side markup into the corresponding table entry. Generic attributes mapped to database table columns will also show a special icon and their value will appear in italics. Such generic attributes cannot be modified, but their value can be deleted.

When the generic attribute is filled, the system checks for database definition integrity and examines all attributes of the selected generic instance. The generic attribute contains the information given by the markup, including which column of the results delivered by a query represents the value. The attribute is either empty or does not hold server-side markup. If the attribute holds markup, the database name and the query ID of the content on the current selection must be the same. This issue must be checked to ensure that result fields come from the same database and the same query. If the system does not check this, it could query nonmatching information.

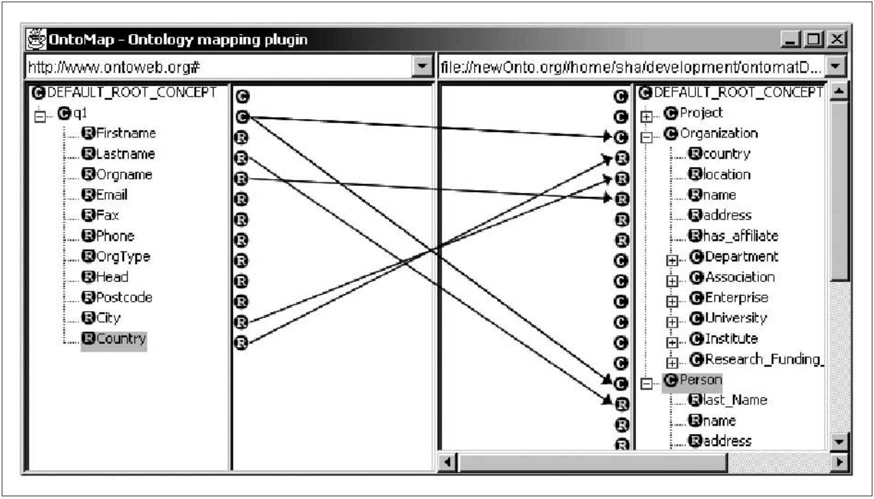


Figure 6. Mapping between the server database (left) and the client ontology (right).

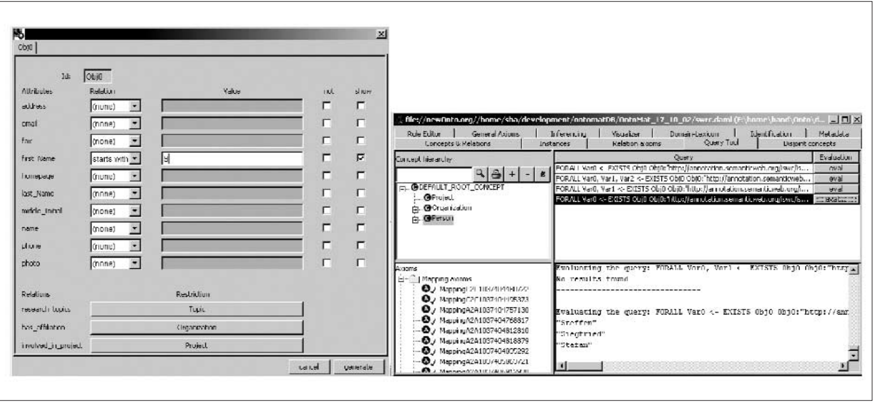


Figure 7. Querying the server database through the client ontology.

Mapping and querying

The results of the annotation represent mapping rules between the database and the client ontology. The annotator publishes the client ontology and the mapping rules derived from annotations. We used the Ontobroker format to publish the mapping rules. The Ontobroker format lets our system use third parties to access and query the database on the basis of the semantic defined in the ontology. The user of this mapping description might be a software agent or a human user. The querying party uses a corresponding tool to visualize the client ontology, investigate the mapping, and compile a query from the client ontology. In our case, we used the OntoEdit plugins OntoMap and OntoQuery.

OntoMap visualizes the database query, the client ontology's structure, and the mapping between them, as Figure 6 shows. The user can control and change the mapping and also create additional mappings.

OntoQuery is a query-by-example user interface. Users create a query by clicking on a concept and selecting the relevant attributes and relationships. The underlying Ontobroker system transforms the ontological query into a corresponding SQL query. Ontobroker uses the mapping descriptions, which are internally represented as F-Logic axioms, to transform the query. The SQL query will be sent as an RPC call to the Web service, where it will be answered in the form of a set of records. The system changes these records back into an ontological representation automatically so that no interaction with the user is necessary.

For example, one user could create a query by selecting the concept **Person**. In the dialog, the user can restrict the search to instances of **Person** starting with the letter "S" in the name (see the left side of Figure 7). The system expresses this ontological query as an F-Logic query so that Ontobroker can evaluate it using the mapping axioms (see the right side of Figure 7). The data migration executes in two separate steps. In the first step, all the required concept instances are created without considering relationships or attributes. The instances are stored together with their identifier. The identifier is translated from the database keys using the identifier pattern. For example, the instance with the name **AIFB** of the concept **Institute**, which is a subconcept of **Organization**, has the identifier www.ontoweb.org/org/OrganizationID=3.

After creating all instances, the system starts computing the values of the instance relationships and attributes. The way the system assigns the values depends on the mapping rules. Because the values of an attribute or a relationship must be computed from both the relational database and the ontology, we generate two queries for each attribute relationship: one SQL query and one Ontobroker query. We invoke each query with an instance key value as a parameter—or a corresponding database key in SQL queries—and return the value of the attribute relationship.

The database communication takes place through bind variables. The system generates the corresponding SQL query, and, if it is the first call, caches the query. A second call would try to use the same database cursor—if it is still available—without parsing the respective SQL statement. Otherwise, it would find an unused cursor and retrieve the results. In this way, the system would maintain efficient access methods for relations and database rules throughout the session.

The deep-annotation technique leaves semantic data in database systems, where it can be handled most effectively. In this way, deep annotation provides a means to map and reuse dynamic data in the Semantic Web with comparatively simple and intuitive tools.

Although we have provided a complete framework and its prototype implementation for deep annotation, there is still a long list of unresolved issues—from the mundane to the far-reaching. So far, we have only considered atomic database fields. For example, you might be able to find a book by searching the entire bibliography reference as a title query. But you might instead be interested in separating this field into, for example, title, publisher, location, and date.

In addition, a content-management system such as Zope (www.zope.org) could provide the means for automatically deriving server-side Web page markup for deep annotation. The database provider could then be freed from any workload while still providing for participation in the Semantic Web. For now, we have built our deep-annotation process on SQL and relational databases. Future schemes could exploit XQuery (www.w3.org/TR/xquery) or an ontology-based query language. And in the future, deep annotations might even link to each other, creating a dynamic interconnected Semantic Web that allows translation between different servers.

Querying the database directly certainly could create problems, such as new possibilities for denial-of-service attacks. In fact, queries such as the ones that involve too many joins over large tables might prove hazardous. Nevertheless, we see this problem as a challenge to be solved by clever schemes for CPU processing time. Ultimately, we believe extending deep-annotation research along these lines might make an intriguing scheme on which a considerable part of the Semantic Web could be built. ■

Acknowledgments

Research for this article was funded by the projects DARPA DAML OntoAgents, EU IST Bizon, and EU IST WonderWeb. We gratefully thank Leo Meyer and Dirk Wenke, Ontoprise, for implementations that contributed to the deep-annotation prototype described in this article.

References

1. S. Handschuh and S. Staab, "Authoring and Annotation of Web Pages in CREAM," *Proc. 11th Int'l World Wide Web Conf.*, ACM Press, 2002, pp. 462–473.
2. M. Vargas-Vera et al., "MnM: Ontology Driven Semiautomatic and Automatic Support for Semantic Markup," *Proc. European Knowledge Acquisition Workshop 2002*, Springer-Verlag, 2002, pp. 379–391.
3. J. Golbeck et al., "New Tools for the Semantic Web," *Proc. European Knowledge Acquisition Workshop 2002*, Springer-Verlag, 2002, pp. 392–400.
4. A. Sahuguet and F. Azavant, "Building Intelligent Web Applications Using Lightweight Wrappers," *Data and Knowledge Eng.*, vol. 3, no. 36, 2001, pp. 283–316.
5. D. Fensel et al., "On2broker: Semantic-Based Access to Information Sources at the WWW," *Proc. World Conf. on the WWW and Internet*, IEEE CS Press, 1999, pp. 366–371

The Authors

Siegfried Handschuh is a researcher at the Institute of Applied Computer Science and Formal Description Methods at the University of Karlsruhe. His research interests include annotations in the Semantic Web and ontology-based applications. He received a degree in information science from the University of Constance. Contact him at Institute AIFB, Univ. of Karlsruhe, 76128 Karlsruhe, Germany; sha@aifb.uni-karlsruhe.de.

Steffen Staab is a lecturer in applied computer science at the University of Karlsruhe. His research interests include computational linguistics, knowledge discovery, knowledge management, ontologies, and the Semantic Web. He received an MSE in computer and information science from the University of Pennsylvania, a PhD in informatics from Freiburg University, and a habilitation in applied informatics from the University of Karlsruhe. Contact him at Ontoprise GmbH, 76131 Karlsruhe, Germany; sst@aifb.uni-karlsruhe.de.

Raphael Volz is a researcher at the Institute of Applied Computer Science and Formal Description Methods at the University of Karlsruhe. His research interests include the intersection of traditional database theory and Semantic Web technologies. He received a degree in computer science from the University of Karlsruhe. Contact him at Institute AIFB, Univ. of Karlsruhe, 76128 Karlsruhe, Germany; rvo@aifb.uni-karlsruhe.de.